Schaal S (2002) Learning robot control. In: Arbib MA (ed) The handbook of brain theory and neural networks, 2nd Edition. MIT Press, Cambridge, MA, pp 983-987

Learning Robot Control

Stefan Schaal

sschaal@usc.edu; http://www-slab.usc.edu
Computer Science and Neuroscience & Kawato Dynamic Brain Project (ERATO/JST)
University of Southern California, 3614 Watt Way–HNB103, Los Angeles, CA 90089-2520

Introduction

Learning robot control, a subclass of the field of learning control, refers to the process of acquiring a sensory-motor control strategy for a particular movement task and movement system by trial and error. Learning control is usually distinguished from adaptive control (see ADAPTIVE CONTROL) in that the learning system is permitted to fail during the process of learning, while adaptive control emphasizes single trial convergence without failure. Thus, learning control resembles the way that humans and animals acquire new movement strategies, while adaptive control is a special case of learning control that fulfills stringent performance constraints, e.g., as needed in life-critical systems like airplanes and industrial robots.

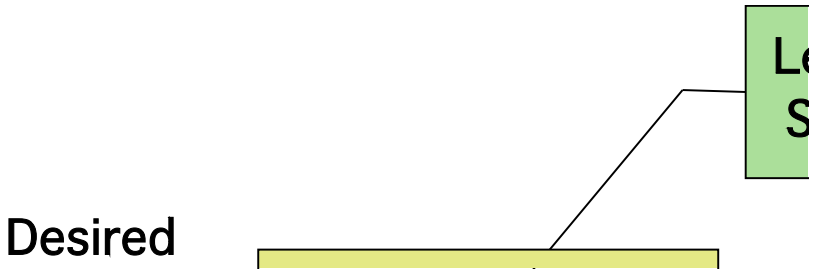

Figure 1: Generic control diagram for learning control

A key question in learning control is what it is that should be learned. In order to address this issue, it is helpful to assume one of the most general frameworks of learning control as originally developed in the middle of the 20th century in the fields of optimization theory, optimal control, and in particular dynamic programming (Bellman, 1957; Dyer & McReynolds, 1970). Here, the goal of learning control was formalized as the need to acquire a task dependent control policy π that maps the continuous valued state vector $\mathbf{x}$ of a control system and its environment, possibly in a time dependent way, to a continuous valued control vector $\mathbf{u}$:

$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t) \tag{1}$$

The parameter vector α contains the problem specific parameters in the policy π that need to be adjusted by the learning system; Figure 1 illustrates the generic

control diagram of learning a control policy. Since the controlled system can generally be expressed as a nonlinear function

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{2}$$

in accordance with standard dynamical systems theory (Strogatz, 1994), the combined system and controller dynamics result in:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \pi(\mathbf{x}, t, \alpha)) \tag{3}$$

Thus, learning control means finding a (usually nonlinear) function π that is adequate for a given desired behavior and movement system. This formal viewpoint allows discussing robot learning in terms of the different methods that have been suggested for the learning of control policies.

Methods of Learning Robot Control

Learning the Control Policy Directly

It is possible to learn the control policy π directly, i.e., without splitting it into subcomponents as explained in later sections. For this purpose, the desired behavior needs to be expressed as an optimization criterion $r(t)$ to be optimized over a certain temporal horizon T, resulting in a cost function

$$J(\mathbf{x}(t)) = \int_{t=0}^{T} r(\mathbf{x}(s), \mathbf{u}(s))ds \quad or \quad J(\mathbf{x}(t)) = \int_{t=0}^{\infty} \frac{1}{\tau} e^{-\frac{s-t}{\tau}} r(\mathbf{x}(s), \mathbf{u}(s))ds \tag{4}$$

The second formulation of equation (4) illustrates the use of an infinite horizon time window by introducing a discount factor that reduces the influence of values of $r(t)$ in the far future. Note that $r(t)$ is usually a function of the state $\mathbf{x}$ and command $\mathbf{u}$ taken in $\mathbf{x}$, i.e., $r(t) = r(\mathbf{x}(t), \mathbf{u}(t))$. Solving such kinds of optimization problems was developed in the framework of dynamic programming (Dyer & McReynolds, 1970) and its recent derivative, reinforcement learning (Sutton & Barto, 1998, see REINFORCEMENT LEARNING). For reinforcement learning, $r(t)$ corresponds to the 'immediate reward', and $J(\mathbf{x}(t))$ is called the 'long-term reward'. For instance, in the classical task of balancing a pole on a finger, the immediate reward could be "+1" at every time step at which balancing was successful, and "-1000" if the pole was dropped; the task goal would be to accumulate maximal long term reward, equivalent to balancing without dropping.

The policy π is acquired with reinforcement learning by, first, learning the optimal function $J(\mathbf{x}(t))$ for every state $\mathbf{x}$, usually by a technique called temporal-difference learning (see REINFORCEMENT LEARNING), and then deducing the policy π as the command $\mathbf{u}$ in every state $\mathbf{x}$ that leads to the best future payoff, i.e.:

$$\mathbf{u} = \max_{\mathbf{u}'}(r(\mathbf{x}(t), \mathbf{u}'(t)) + J(\mathbf{x}(t+1))), \text{ where } \mathbf{x}(t+1) = \mathbf{x}(t) + f(\mathbf{x}(t), \mathbf{u}'(t))\Delta t \tag{5}$$

where Δt is an arbitrarily chosen constant time step for sampling the system's behavior. Many variations of reinforcement learning exist, including methods

that avoid estimating the optimization function $J(\mathbf{x}(t))$ (see REINFORCEMENT LEARNING).

Learning the Control Policy in a Modular Way

Theoretically, the techniques of reinforcement learning and dynamic programming would be able to solve any robot learning problem that can be formulated as sketched in the previous section. Practically, however, this is not true since reinforcement learning requires a large amount of exploration of all actions and states for proper convergence of learning as well as appropriate representations for the function $J(\mathbf{x}(t))$. Traditionally, $J(\mathbf{x}(t))$ needs to be represented as a lookup table, i.e., for every state $\mathbf{x}$ a specific table cell holds the appropriate value, $J(\mathbf{x}(t))$. For continuous valued states, discretization of the individual dimensions is needed. For high dimensional systems, this strategy leads to an explosion of lookup table cells, e.g., for a 30 dimensional movement system where each dimension is just split into two cells, an astronomical number of 2^{30} cells would be required. Exploring all these cells with a real robot would take forever, and even in computer simulations such problems are not tractable. Newer approaches employ special neural networks for learning and representing $J(\mathbf{x}(t))$ (e.g., Sutton & Barto, 1998), but problems with high-dimensional movement systems remain daunting.

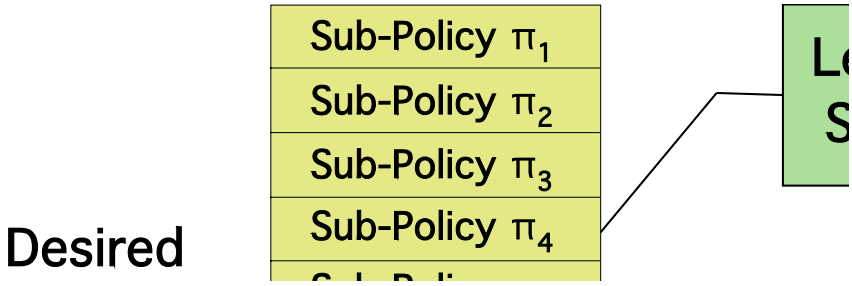

Figure 2: Learning control with sub-policies

A possible way to reduce the computational complexity of learning a control policy comes from modularizing the policy (Figure 2). Instead of learning the entire policy in one big representation, one could try to learn sub-policies that have reduced complexity and, subsequently, build the complete policy out of such sub-policies. This approach is also appealing from the viewpoint of learning multiple tasks: some of the sub-policies may be re-used in another task such that learning new tasks should be strongly facilitated.

Motor control with sub-policies has been explored in various fields, for instance, under the names of schema theory (see SCHEMA THEORY), behavior based robotics (see REACTIVE ROBOTIC SYSTEMS), pattern generators (see MOTOR PATTERN GENERATION), and movement primitives (Schaal, 1999). Robot learning with such modular control systems, however, is still in its infancy. Reinforcement learning has recently begun to formulate a principled approach to this problem (Sutton, Precup, & Singh, 1999). Another route of investigating modular robot learning comes from formulating sub-policies as nonlinear dynamical systems

(Mussa-Ivaldi & Bizzi, 1997, Schaal & Sternad, 1998). However, all this research is still of preliminary nature and will take some more time before it is applicable to complex robot learning problems.

Indirect Learning of Control Policies

The previous sections assumed that motor commands are directly generated based on the information of the state of the world $\mathbf{x}$, i.e. from the policy function π. For many movement systems, however, such a *direct* control strategy is not advantageous since it fails to re-use modules in the policy that are common across multiple tasks. This view suggests that, in addition to a modularization of motor control and learning in form of a mixture of simpler policies (Figure 2), modularization can also be achieved in terms of a functional structuring within each control policy. A typical example is to organize the control policy into several processing stages, as illustrated in Figure 3 and also discussed as *indirect* control in MOTOR CONTROL, BIOLOGICAL AND THEORETICAL. Most commonly, the policy is decomposed into a planning and an execution stage, a strategy that is typical for most robot controllers but also likely to be used in motor control of primates. Planning generates a desired *kinematic* trajectory, i.e., a prescribed way of how the state of the movement system is supposed to change in order to achieve the task goal. Execution transforms the plan into appropriate motor commands.

Separating planning and execution is highly advantageous. For instance, in reaching movements for a target, a *direct* approach to robot learning would require to learn a new policy for every target location—the desired behavior is to minimize the distance between the hand and the target, and due to the complex dynamics of an arm, different target locations require very different motor commands for reaching. In contrast, an indirect control approach only requires learning the movement execution module, usually in form of an inverse model (see below). The execution module is valid for any target location. For simplicity, movement plans can be kept the same for every target location, e.g., a straight line between the target and the start position of the arm with a bell-shaped velocity profile—planning such movement kinematics requires only one-time learning of the robot kinematics model and using standard kinematic planning algorithms (Sciavicco & Siciliano, 1996) that can easily cope with any reachable target location. Thus, after the execution module has been acquired, reaching is a largely solved problem, no matter where the target is.

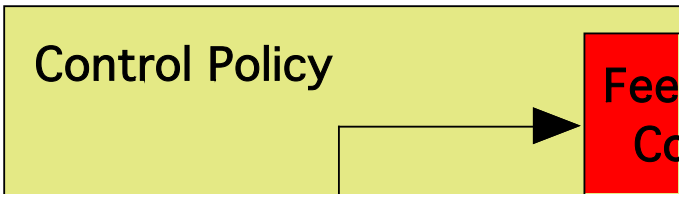

Figure 3: Learning control with functional decomposition

Depending on the task, planning can take place in external kinematic variables, e.g., Cartesian or endeffector space, or in internal kinematic variables, e.g., joint space for a human-like arm. If the planning space does not coincide with the space where motor commands are issued, coordinate transformations are required to map the external motor plan into intrinsic coordinates. This problem is typically discussed as the inverse kinematics problem (see ROBOT CONTROL) of robot control, a problem that equally needs to be addressed by biological movement systems.

To transform kinematic plans into motor commands, standard methods from control theory can be employed (e.g., Sciavicco & Siciliano, 1996). Figure 3 illustrates a typical example that uses a feedforward/feedback mechanism—called "computed torque controller"—that enjoys popularity in both robotic systems as well as models of biological motor control (Jordan, 1996). The feedback controller is of classical Proportional-Derivative (PD) type, while the feedforward controller contains an inverse dynamics model of the movement system (see CEREBELLUM AND MOTOR CONTROL).

From the point of robot learning, functional modularity also decomposes the learning problem into several independent learning problems. The modules of the execution stage can be learned with supervised learning techniques (see below and also CEREBELLUM AND MOTOR CONTROL). For various types of movements, kinematic movement plans can be highly stereotyped, as described in the reaching example above, such that no learning is required for planning. For complex movements, e.g., a tennis serve, planning requires more sophistication, and the same reinforcement learning methods of *direct* control can be applied—the only difference is that motor commands $\mathbf{u}$ are replaced with a desired change in trajectory $\dot{\mathbf{x}}_d$. Applying reinforcement learning to kinematic planning is of reduced complexity than solving the complete *direct* control problem since the highly nonlinear transformation from kinematic plans to motor commands does not need to be acquired anymore, but still it remains an open research problem how to perform reinforcement learning for high dimensional movement systems.

Imitation Learning

A topic in robot learning that has recently received increasing attention is that of imitation learning. The idea of imitation learning is intuitively simple: a student watches the performance of a teacher, and, subsequently, uses the demonstrated movement as a seed to start his/her own movement. The ability to learn from imitation has a profound impact on how quickly new skills can be acquired (Schaal, 1999). From the viewpoint of learning theory, imitation can be conceived as a method to bias learning towards a particular solution, i.e., that of the teacher. Motor learning proceeds afterwards as described in the other sections of this chapter. However, not every representation for motor learning is equally suited to be biased by imitation (Schaal, 1997). For instance, a robot using *direct* control can hardly profit from a demonstration as motor commands are not perceivable, but a robot using *indirect* control could extract a first kinematic plan from the demonstration and use it for starting its own learning. The ability of imitation thus imposes interesting constraints on the structure of a learning system for motor learning.

Learning of Motor Control Components

Whether direct or indirect control is employed in a motor learning problem, the core of the learning system usually requires methods of supervised learning of regression problems, called function approximation in the neural network and statistical learning literature. Function approximation is concerned with approximating a nonlinear function $\mathbf{y}=f(\mathbf{x})$ from noisy data, drawn according to the data generating model:

$$\mathbf{y} = f(\mathbf{x}) + \varepsilon \quad \text{where} \quad \mathbf{x} \in \Re^n, \mathbf{y} \in \Re^m, E\{\varepsilon\} = 0 \tag{6}$$

i.e., $\mathbf{x}$ is an n-dimensional continuous valued vector, $\mathbf{y}$ is an m-dimensional continuous valued vector, and ε a mean-zero random "noise" variable. By comparing the generic form of a policy in Equation (1) or a dynamics model in Equation (2) with (6), it is apparent that learning such functions falls into the framework of function approximation.

Neural Network Approaches to Function Approximation

Many different methods of function approximation exist in the literature (see LEARNING AND STATISTICAL INFERENCE). For the purpose of this chapter, it is sufficient to classify all these algorithms into two categories, spatially localized (*local*) algorithms, and spatially non-localized (*global*) algorithms. The power of learning in neural networks comes from the nonlinear activation functions that are employed in the hidden units of the neural network. *Global* algorithms use nonlinear activation functions that are not limited to a finite domain in the input space ($\mathbf{x}$-space) of the function. The prototypical example is the sigmoid function in Figure 4a that outputs a value of roughly one for any input greater than about one. In contrast, *local* algorithms make use of nonlinear activation functions that are different from zero only in a restricted input domain—the Gaussian function

in Figure 4b exemplifies this class of functions. Despite that both *local* and *global* algorithms are theoretically capable of approximating arbitrarily complex non-linear functions, the learning speed, convergence, and applicability to high-dimensional learning problems differ significantly (Schaal & Atkeson, 1998).

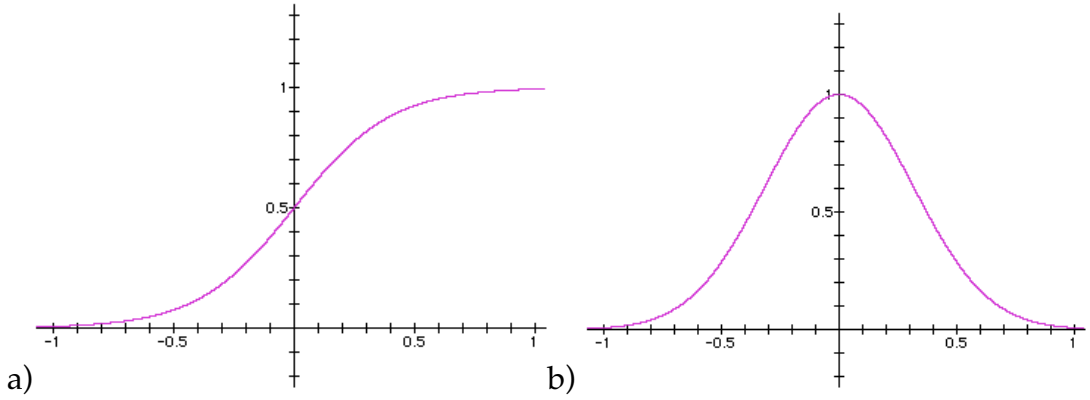

Figure 4: Nonlinear activation functions used in neural networks: a) the sigmoid function, a spatially global function, b) the Gaussian function, a spatially localized function.

A metaphor of how *global* algorithms approximate nonlinear functions is to conceive of them as an octopus whose tentacles stretch out and span the complex surface described by $\mathbf{y}=f(\mathbf{x})$, except that the tentacles live in an n-dimensional space. *Global* algorithms can work quite well for problems with many input dimensions since their non-local activation function (i.e., their "tentacles") can span even huge spaces quite efficiently. But *global* algorithms usually require very careful training procedures such that the "tentacles" learn how to stretch appropriately into all directions. In particular, if at some point of training, data is only provided for a restricted area in input space, the "tentacles" may focus too much on approximating this area and, while doing so, forget maintaining the "tentacle posture" in previously learned areas—a phenomenon called catastrophic interference (Schaal & Atkeson, 1998). Catastrophic interference is particularly pronounced in incremental learning problems, where training data comes point after point and can only be used once for updating the algorithm—unfortunately this is the typical scenario in robot learning. Together with the problem on how to select the right number of hidden units (i.e., the right number of tentacles), it becomes quite complicated to train *global* algorithms for high dimensional robot learning problems.

In contrast, *local* learning algorithms have quite different characteristics. The metaphor for *local* learning is simply that they approximate the complex regression surface with the help of small local patches, for instance locally constant or locally linear functions (Atkeson, Moore, & Schaal, 1997). Problems of how many patches need to be allocated, where to place them, how large they should be in input space, and how to learn them incrementally have largely been solved (Schaal & Atkeson, 1998). The biggest problem of *local* algorithms is the curse of dimensionality, i.e., the exponential explosion of the number of patches that are needed in high dimensional input spaces. For instance, assume that we want to

divide every input dimension of a function approximation problem into ten local regions. For two input dimensions, this strategy would result in 10^2 local regions, for three inputs in 10^3, and for n inputs in 10^n regions. Even for only 12 input dimension, this number reaches the number of neurons in the human brain. The only way to avoid this problem is to make the patches larger, but then the quality of function approximation becomes unacceptably inaccurate. There is theoretically no way out of the curse of dimensionality—but empirically, it turns out not to be a problem. The example above that demonstrates the curse of dimensionality can equally be turned around in our favor: how long would it take a robot system to generate all the data points to fill these big spaces? E.g., collecting 10^{12} data points at 100Hz sampling frequency would take more than 300 years of uninterrupted movement! Thus, no actual robot will ever be able to generate enough data to fill these huge spaces. This argument triggers a most important question: what kind of data distributions are actually realized by robotic (or biological) movement systems? Vijayakumar and Schaal (2002) found that distribution had only about 4-6 dimensions *locally* in a robot learning problem that had 21 input dimensions, a finding that was also duplicated in other robot learning domains (Vlassis, Motomura, & Krose, 2002). *Local* learning can exploit this property by using techniques of local dimensionality reduction and can thus learn efficiently even in very high dimensional spaces (Vijayakumar et al., 2002). Thus, for the time being, *local* learning algorithms seem to be better suited for robot learning.

Specific Function Approximation Problems in Robot Learning

Applying function approximation to problems of robot learning requires a few more considerations. The easiest applications are those of straightforward supervised learning, i.e., where a teacher signal $\mathbf{y}$ is directly available for every training point $\mathbf{x}$. For example, learning a dynamics model of the form of Equation (2) falls usually into this category if the inputs $\mathbf{x}$ and $\mathbf{u}$, and the output $\dot{\mathbf{x}}$ can be measured directly from sensors. Many other problems of ROBOT CONTROL are of a similar nature.

Learning becomes more challenging when instead of the teacher signal only an error signal is provided, and the error signal is just approximate. Assume we have such an error signal $\mathbf{e}$ when the network predicted a particular $\hat{\mathbf{y}}$ for a given input $\mathbf{x}$. From this information, we can create a teacher signal $\mathbf{y} = \hat{\mathbf{y}} + \mathbf{e}$ and train the network with this target. However, if $\mathbf{e}$ was only approximate, $\mathbf{y}$ is not the true target, and later on during learning another (hopefully more accurate) teacher signal may be formed for training the network. Thus, learning proceeds with "moving targets", which is called a nonstationary learning problem. For such learning tasks, neural networks need to have an appropriate amount of plasticity in order to keep on changing until the targets become correct. On the other hand, it is also important that the network converges at some point and averages out the noise in the data, i.e., that the network does not have too much plasticity. Finding appropriate neural networks that have the right amount of

plasticity-stability tradeoff is a non-trivial problem, and so far, heuristic solutions dominate the literature.

Nonstationary learning problems are unfortunately quite common in robot learning. Learning the optimization function $J(\mathbf{x}(t))$ in reinforcement learning is one typical example since the temporal difference algorithm (Sutton & Barto, 1998) can only provide approximate errors. Other examples include feedback error learning and learning with distal teachers (Jordan, 1996). Both of these methods address the problem that in learning control, we usually only receive errors in sensory variables, e.g., positions and velocities, but what is needed to train a control policy is an error in motor commands. Thus, feedback error learning creates an approximate motor command error by using the output of a linear feedback controller as the error signal. Learning with distal teachers accomplishes essentially the same goal, except that it employs a learned forward model to map an error in sensory space to an approximate motor error.

Applications

While the theoretical development of learning control has progressed significantly in recent years, applications in actual robots have remained rather sparse due to the significant computational burden of most learning algorithms and the real-time constraints of actual robots. Reinforcement learning in actual robots remains largely infeasible, and only few examples exist in simplified setups (e.g., see references in Atkeson et al., 1997; Schaal, 1999; Sutton & Barto, 1998). Learning of internal models for robot control has found increasingly more widespread application due to significant advances in the computational efficiency of supervised learning algorithms (e.g., see references in Atkeson et al., 1997; Vijayakumar et al., 2002; Vlassis et al., 2002).

Discussion

Robot learning is a surprisingly complex problem. It needs to address how to learn from (possibly delayed) rewards, how to deal with very high-dimensional learning problems, how to use efficient function approximators, and how to embed all the elements in a control system with real-time and robust performance. A further difference to many other learning tasks is that in robot learning the training data is generated by the movement system itself. Efficient data generation, i.e., exploration of the world, will result in fast learning, while inefficient exploration can even prevent successful learning all together (see REINFORCEMENT LEARNING). Given the fact that only very few robots in the world are equipped with learning capabilities yet, it becomes obvious that research on robot learning is still in an early stage of its development.

References

*Atkeson, C. G., Moore, A. W., & Schaal, S. (1997). Locally weighted learning for control. *Artificial Intelligence Review*, **11,** 75-113.

Bellman, R. (1957). *Dynamic programming*. Princeton, N.J.: Princeton University Press.

Dyer, P., & McReynolds, S. R. (1970). *The computation and theory of optimal control*. New York: Academic Press.

*Jordan, M. I. (1996). Computational aspects of motor control and motor learning. *In* H. Heuer, & S. W. Keele (Eds.), *Handbook of perception and action*. New York: Academic Press.

Mussa-Ivaldi, F. A., & Bizzi, E. (1997). Learning Newtonian mechanics. *In* P. Morasso, & V. Sanguineti (Eds.), *Self-organization, Computational Maps, and Motor Control* (pp. 491-501). Amsterdam: Elsevier.

Schaal, S. (1997). Learning from demonstration. *In* M. C. Mozer, M. Jordan, & T. Petsche (Eds.), *Advances in Neural Information Processing Systems 9* (pp. 1040-1046). Cambridge, MA: MIT Press.

*Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, **3,** 233-242.

Schaal, S., & Atkeson, C. G. (1998). Constructive incremental learning from only local information. *Neural Comput.*, **10,** 2047-2084.

Schaal, S., & Sternad, D. (1998). Programmable pattern generators, *3rd International Conference on Computational Intelligence in Neuroscience* (pp. 48-51). Research Triangle Park, NC.

Sciavicco, L., & Siciliano, B. (1996). *Modeling and control of robot manipulators*. New York: MacGraw-Hill.

Strogatz, S. H. (1994). *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. Reading, MA: Addison-Wesley.

*Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning : An introduction*. Cambridge: MIT Press.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, **112,** 181-211.

Vijayakumar, S., D'Souza, A., Shibata, T., Conradt, J., & Schaal, S. (2002). Statistical learning for humanoid robots. *Autonomous Robots*, **12,** 59-72.

Vlassis, N., Motomura, Y., & Krose, B. (2002). Supervised dimension reduction of intrinsically low-dimensional data. *Neural Comput.*, **14,** 191-215.